实验7 取指令及指令译码实验

一、实验目的

- 掌握指令存储器、PC与IR的设计方法；
- 掌握CPU取指令操作与指令译码的方法和过程，掌握指令译码器的设计方法；
- 理解RISC-V立即数的生成与扩展方法，掌握立即数拼接与扩展器的设计。

二、实验内容与原理

- **实验内容：**
  - 设计一个64×32位的只读存储器作为指令存储器；
  - 设计程序计数器PC和指令寄存器IR，实现取指令操作；
  - 设计一个指令译码器，完成指令译码、立即数拼接与扩展操作。

1、指令存储器IM与取指令
2、指令初级译码器ID1
3、立即数拼接与扩展器ImmU

1、指令存储器IM与取指令

两种存储器结构

（1）冯·诺依曼体系结构(普林斯顿结构)

- 将程序指令和数据存储在同一个存储器
- 缺点：信息流传输成为限制计算机性能的瓶颈，影响了数据处理速度的提高

（2）哈佛结构

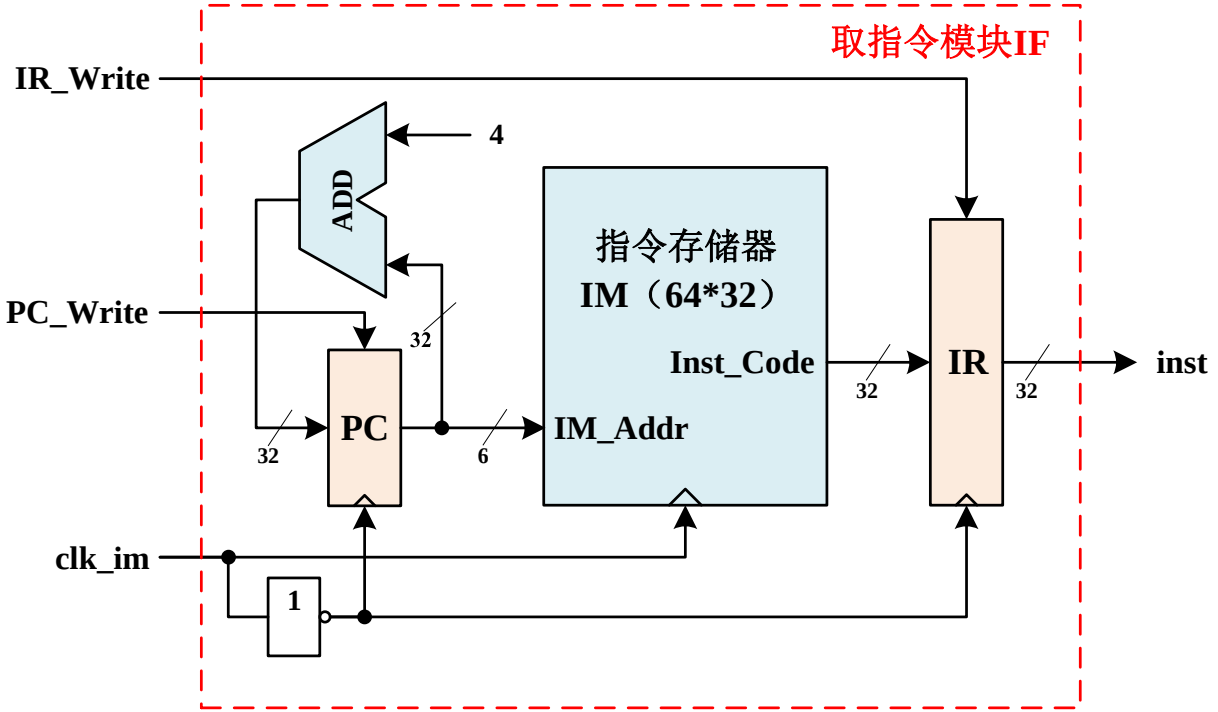
- 将程序的指令和数据分别存储在独立的存储器中
- CPU可以同时读取指令和数据，提高了数据吞吐率；缺点是结构复杂。

本实验的存储器结构

- RISC-V的指令集规范：对存储器结构并没有具体描述
- 本教材设计的模型机：采用哈佛结构设计存储器模块
- 本实验需要设计：一个只读存储器ROM作为指令存储器，用来单独存储程序

1、指令存储器IM与取指令

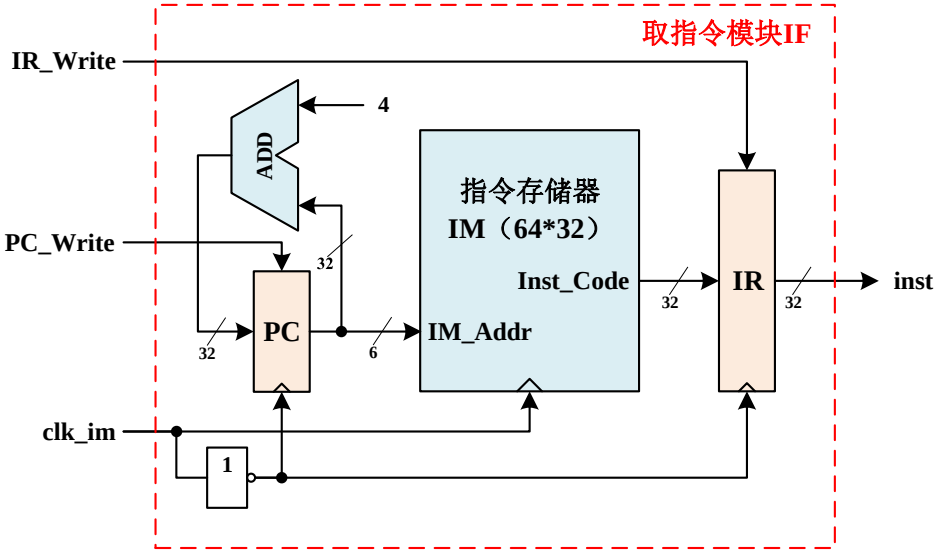
- **取指令模块**
  - （1）指令存储器
  - （2）程序计数器PC
  - （3）PC自增加法器
  - （4）指令寄存器IR



1、指令存储器IM与取指令

（1）指令存储器

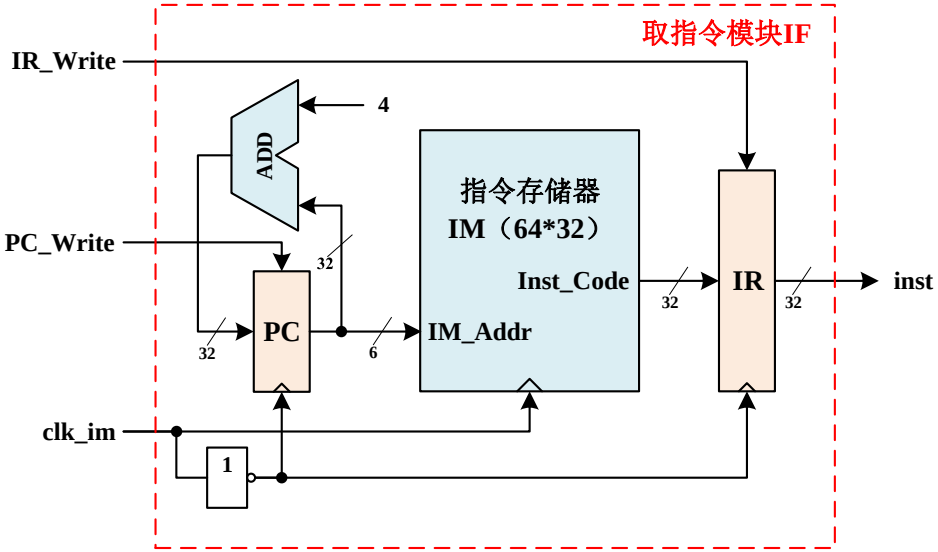
- 只读存储器，只提供读权限，不需要写使能信号。
- 宽度：32位
- 字单元： 64个
- 字地址： 6位
- 字节地址： 8位
- 实现方式：使用Memory IP核生成



1、指令存储器IM与取指令

（2）程序计数器PC

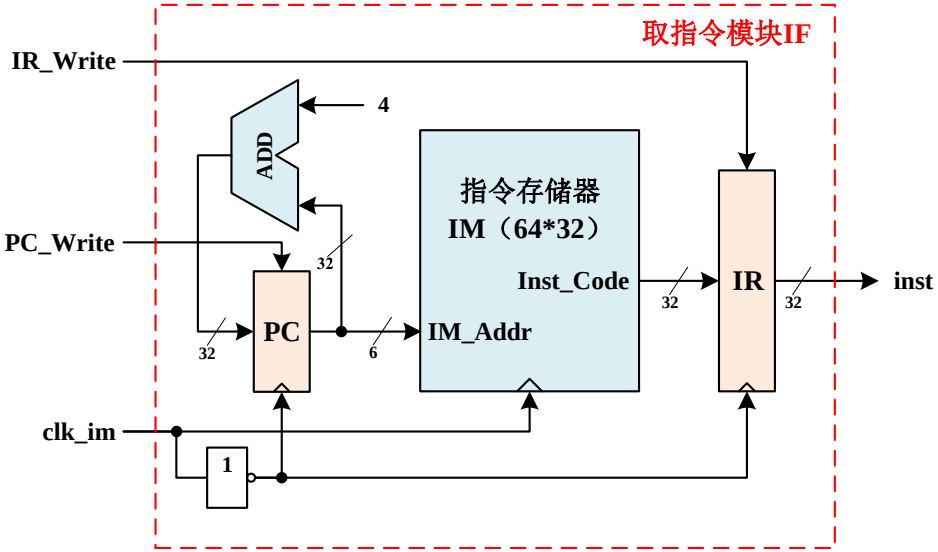
- 存放32位的指令地址，提供给指令存储器来执行取指令操作。
- 低8位作为指令存储器的字节地址， PC[7:2]作为字地址。
- 使用带清零端和置数端的D寄存器实现



1、指令存储器IM与取指令

（3）PC自增加法器

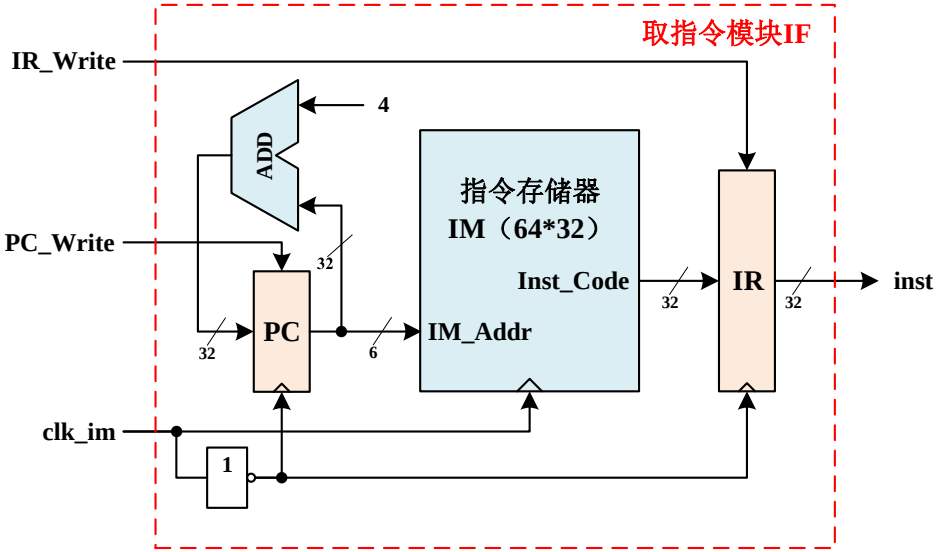
- 用于完成PC的自增操作
- 每次取指令后，PC自增4
- Verilog实现：
  - PC：reg类型变量
  - 自增4加法器： PC+4



1、指令存储器IM与取指令

（4）指令寄存器IR

- 存放根据PC从主存读出的单元内容，为32位的指令代码

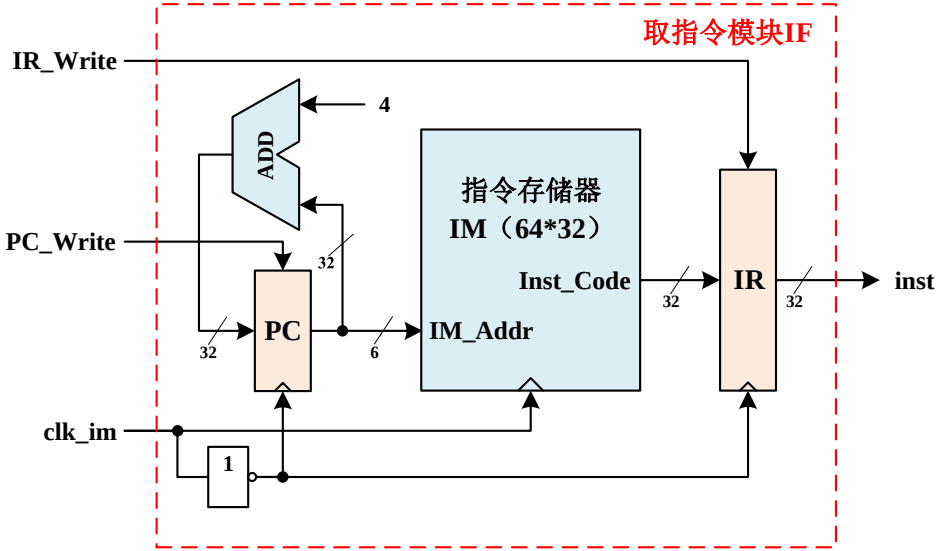


- 取指令操作：根据PC内容到指令存储器中取出指令机器码存入IR，然后PC自增4。

即：Mem[PC]→IR，PC+4→PC

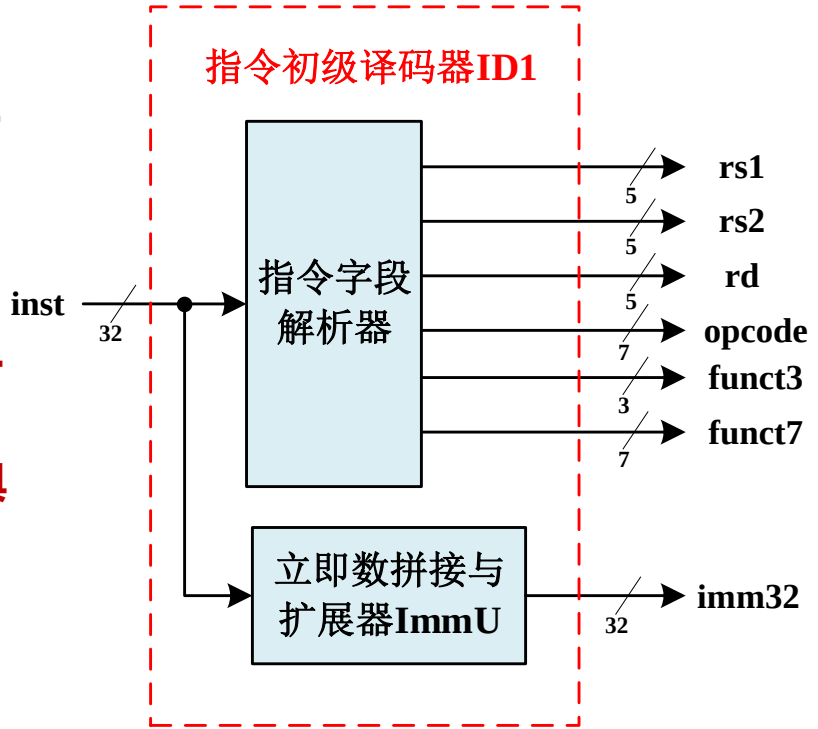
1、指令存储器IM与取指令

- **时序控制：**
  - 目标：在指令存储器读出指令后，PC再置入新的PC+4的值
  - clk上跳沿：根据PC读指令
  - clk下跳沿：将指令打入IR，更新PC值
  - 发送的控制信号：IR_Write，PC_Write



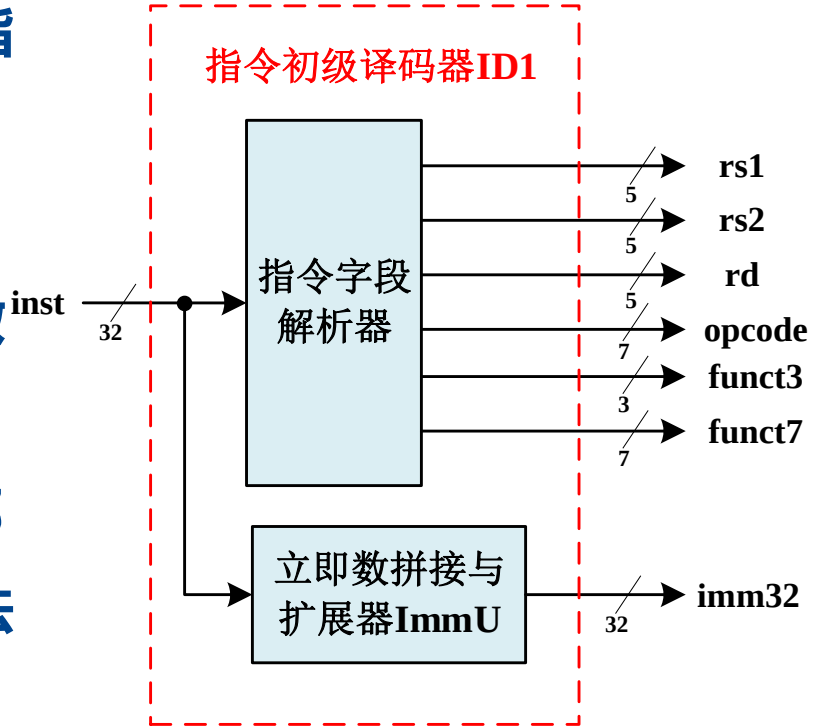
2、指令初级译码器ID1

- **作用：对当前指令进行解析，分辨出是哪一条指令，并识别出操作数及其寻址方式，然后发送控制信号**
- **字段说明：**
  - **rs1、rs2、rd：传递给寄存器堆，作为三个端口的地址；**
  - **opcode、funct3、funct7：用于识别具体的指令；**
  - **imm、shamt：生成立即数，由立即数拼接与扩展器模块ImmU部件实现。**



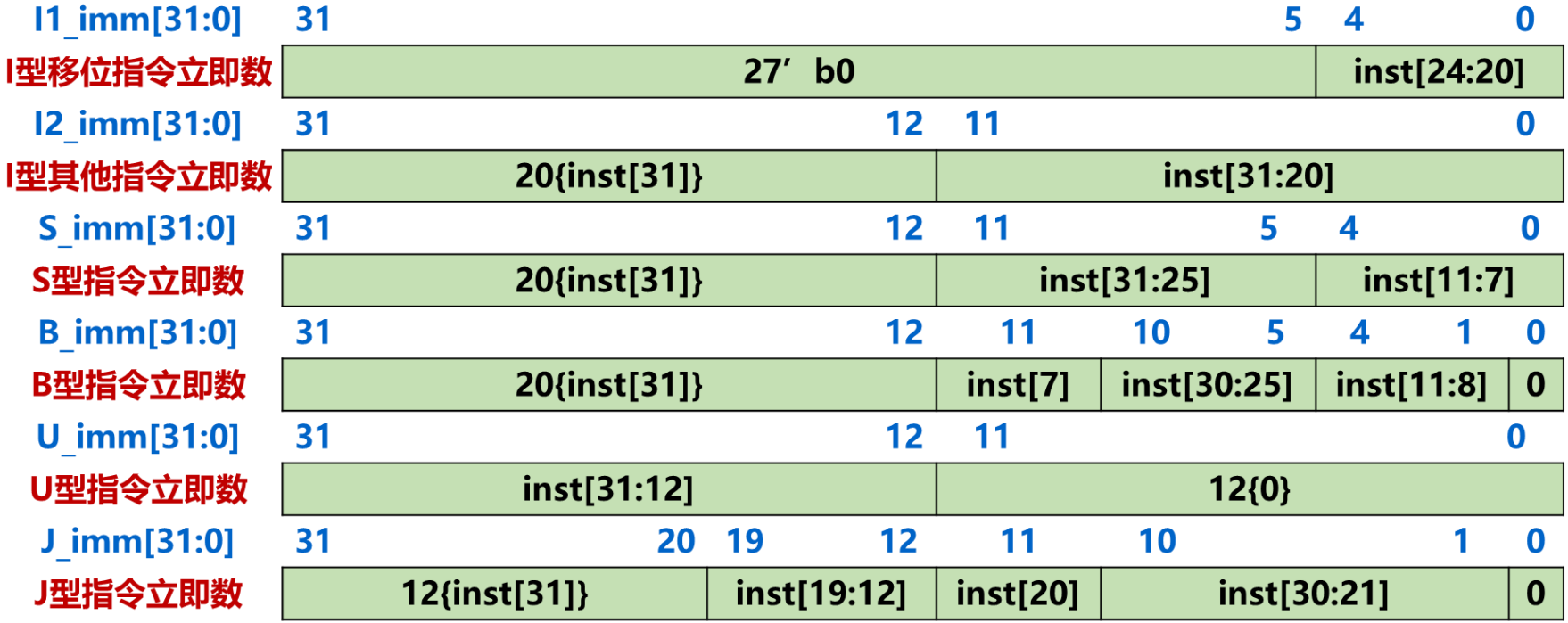
3、立即数拼接与扩展器ImmU

- **作用：对32位指令码译码，识别出指令格式，然后拼接组合立即数字段imm，并扩展产生32位的立即数。**
- **RISC-V的5种指令格式都含有立即数字段imm或者shamt**
- **立即数字段：位数、位置以及位序都各不相同，扩展到32位立即数的方法也有差别**



3、立即数拼接与扩展器ImmU

（1）5种指令格式生成立即数的方法



3、立即数拼接与扩展器ImmU

（2）依据指令格式选择对应的输出（imm32）

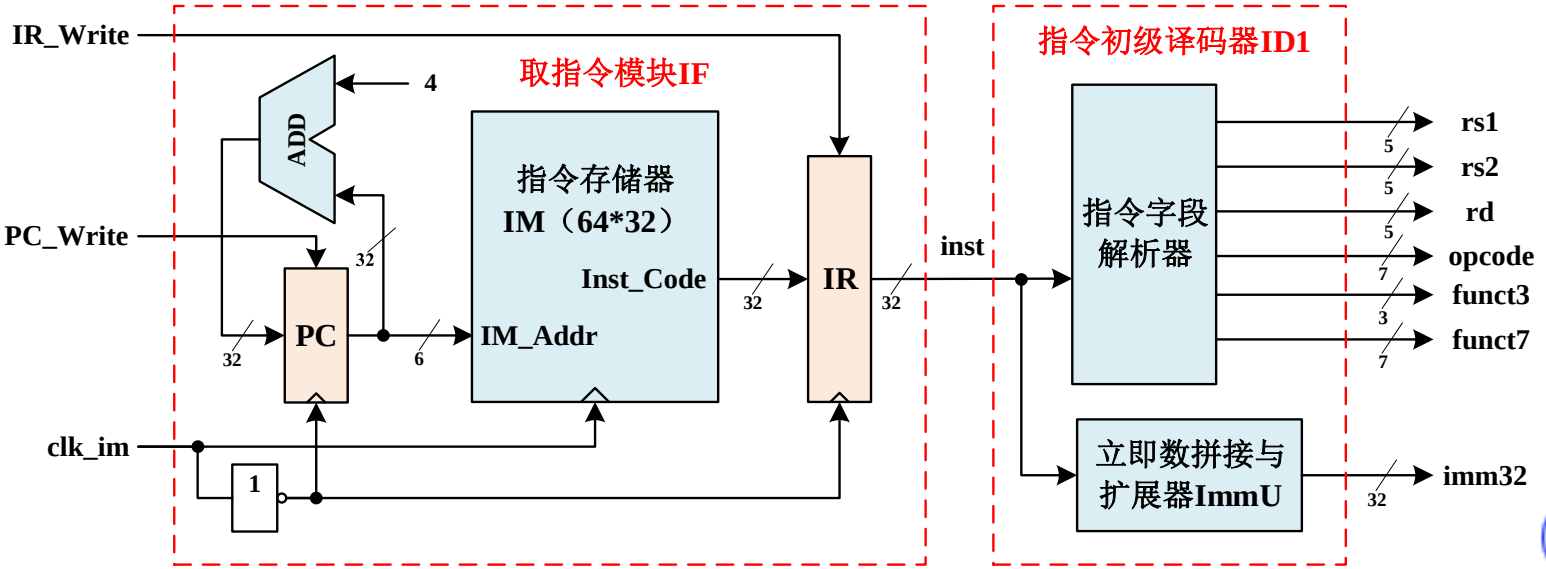
- 指令格式inst_Type可以用独热码或者二进制对其编码；每一种指令格式，都由opcode指定。
- 例：B型指令格式编码：

```
define OP_BRANCH   7‘b1100011          // B型指令格式的操作码

if (opcode == OP_BRANCH)

      inst_Type = 3‘b011;               // B型指令格式编码
```

3、立即数拼接与扩展器ImmU

（3）连接取指令模块和初级译码器

取指令模块是时序逻辑电路，而指令译码是组合逻辑电路。



三、实验要求

1. 编写一个包含**所有指令格式**的测试程序，或者使用实验6的汇编程序，用其构造一个COE文件。
2. 使用Vivado或者ISE的**Memory IP核**，新建一个**64×32**位只读存储器作为**指令存储器IM**，并关联上述COE文件，初始化其内容。
3. 构造**PC**和**IR**寄存器，并与上述指令存储器**IM**连接，构成**取指令模块IF**；对取指令模块进行仿真，确保能**正确取出指令**，并且**PC自增**。
4. 编写指令译码模块**ID1**，包括**立即数拼接与扩展器ImmU**；对ID1模块进行**仿真**，确保指令译码和产生的立即数正确。
5. 连接取指令模块**IF**和指令译码模块**ID1**。

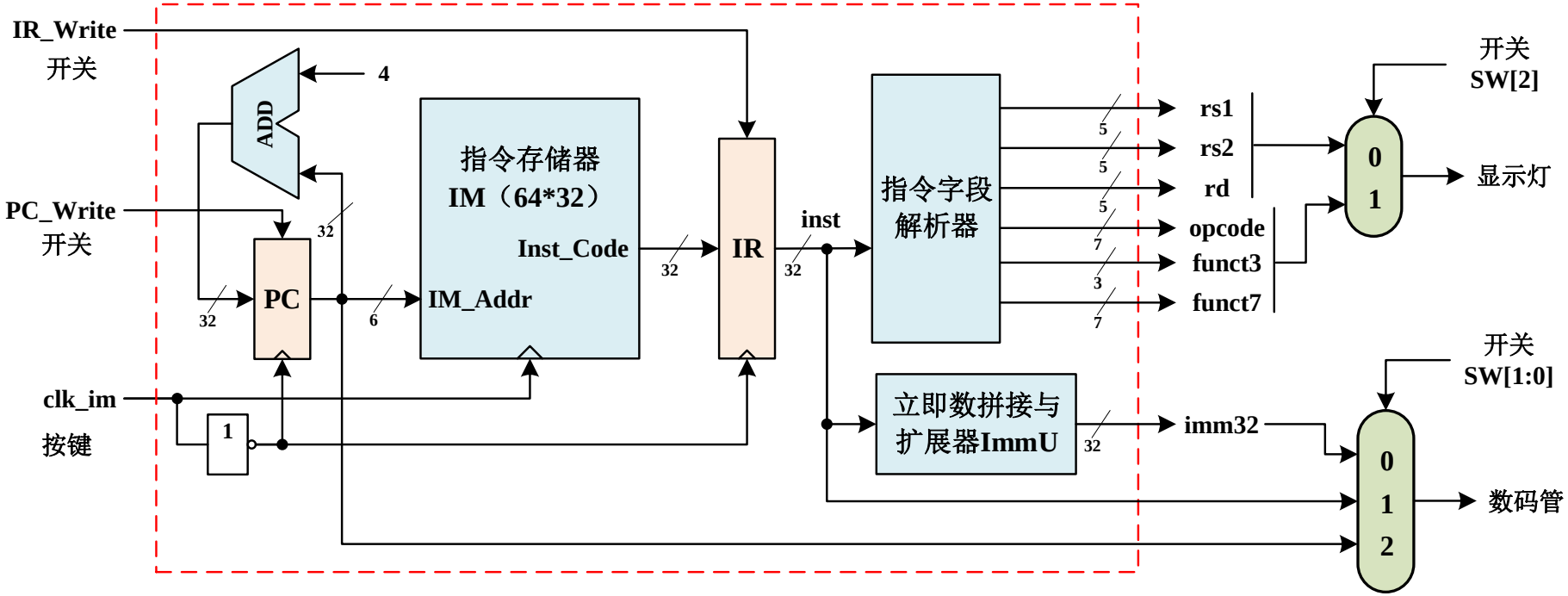
三、实验要求

6. 针对使用的实验板卡，设计取指令与指令译码模块的**板级验证**实验方案，编写**顶层测试**模块。一个示例配置如下：

信号类型	信号名称	配置设备管脚
输入	PC_Write	逻辑开关
	IR_Write	逻辑开关
时钟	rst_n	按键
	clk_im	按键
输出	imm32[31:0]	8个数码管
	rs1[4:0]	5个LED灯
	rs2[4:0]	5个LED灯
	rd[4:0]	5个LED灯
	opcode[6:0]	7个LED灯
	funct3[2:0]	3个LED灯
	funct7[6:0]	7个LED灯

三、实验要求

■ 另一种板级验证方案



三、实验要求

7. 复位后，按clk_im按键，执行取指令操作，验证读出的指令代码是否和COE文件一致，生成的立即数是否正确，解析的各字段是否正确，将实验结果记录到下表中。

PC	IR	COE文件中指令代码	汇编指令	立即数imm32	解析字段是否正确

三、实验要求

8. 实验内容包括：

- 对仿真结果进行分析；
- 描述你设计的板级验证实验方案、模块结构与连接；
- 说明你的板级操作过程；
- 记录板级实验结果。
- 针对以下有待验证的实验问题，给出证据与分析，得到有效结论：

  （1）PC从0号单元开始，每次+4取指令；

  （2）每次读出的指令与指令存储器的初始化内容一致；

  （3）读出的指令，各字段正确解析正确，立即数扩展正确。
- 请力所能及回答或实践本实验的“思考与探索”部分。

四、实验步骤

1. 编写一个包含所有指令格式的汇编程序exp7_test.s，并进行汇编与反汇编；新建一个exp7_test.coe文件，将反汇编文件中的32位机器指令代码逐个拷贝到exp7_test.coe文件中，作为指令存储器的初始化数据。或者：新建一个acc.coe或move.coe文件，然后打开实验6的acc.s或move.s的反汇编文件，将机器指令代码填入COE文件。
2. 新建一个工程，通过IP核生成向导创建一个64×32位的单端口ROM存储器，步骤同实验4，但是Memory Type选择Single Port ROM类型即可；初始化文件选择前述编辑的COE文件。

四、实验步骤

3. 编写PC寄存器模块和IR寄存器模块：复位时清零；打入脉冲来临时，如果写信号有效则执行打入操作。
4. 生成ROM存储器的实例作为指令存储器IM模块，并与上述PC和IR模块连接，构成取指令模块IF。
5. 编写测试代码，对取指令模块进行仿真，清零后，从0号单元开始顺序取指令：当PC_Write和IR_Write保持为1时，每来一个clk_im，就从inst输出一条指令，并且PC+4。
6. 编写指令译码模块ID1，对字段进行解析；对指令格式进行译码识别；编写立即数的拼接与扩展模块ImmU。

四、实验步骤

7. 编写测试代码，对ID1模块进行仿真，确保立即数拼接与扩展正确、确保指令字段解析正确。
8. 构造取指令与译码模块，调用取指令模块IF和指令译码模块ID1，并连接它们。
9. 设计取指令与译码模块的板级验证实验方案，然后据此编写一个顶层测试模块。
10. 新建管脚约束文件，进行相应的引脚配置。
11. 生成*.bit文件，下载到实验设备的FPGA芯片中。

四、实验步骤

12. **板级实验**：按照你所设计的实验方案，操作**输入设备**、观察**输出设备**，预期的验证操作如下：

- 按**rst_n**按键，将**PC**和**IR**清零，接下来从**0号单元**开始取指令；
- 拨开关**PC_Write =1**、**IR_Write=1**，按动时钟键**clk_im**，每按一下，就取出一条指令；
- 观察输出设备上读出的指令是否**按序读出**，代码是否与初始化的coe文件中的**指令码一致**，产生的**立即数**是否正确，并记录到表中。

五、思考与探索（至少完成1道）

1. 在复位后，第一次按动clk_im按钮，你的程序读出的指令是哪个单元的？0号单元还是4号单元的指令？分析为什么？如果要求在复位后的第一个clk_im来临时，读出的是0号单元的指令，若没有实现，尝试修改程序实现。（提示：请关注rst_n按键和clk_im按键的有效边沿）
2. 你的实验能同时观察PC和IR（即inst）的值吗？如果可以，请分析指令地址和指令代码是否匹配？如果不匹配，请分析原因。
3. 你在板级验证中，每按一下clk_im按键，指令是否按序一条一条读出？PC是否+4且只+4？你是如何判断的？分析为什么会出现不正常情况，提出解决办法。

五、思考与探索（至少完成1道）

4. 在识别指令格式时，你的程序中，I型格式的指令操作码有几个？你是如何区分I型移位指令和I型其他指令的？
5. 请问你的立即数拼接与扩展模块，对于移位指令slli、srli和srai是否正确？请仿真时，测试一下。如果不正确，请尝试修改，直至正确。
6. 说说你在实验中碰到了哪些问题，你是如何解决的？